1990

Affix agreement in register vector grammar

Edward James Labuda
Lehigh University

Affix Agreement in Register Vector Grammar

by

Edward James Labuda

A Thesis

Presented to the Graduate Committee

of Lehigh University

in candidacy for the degree of

Master of Science in Computer Science

Lehigh University

1990

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

date: 5/11/90

Advisor in charge

CSEE Department Chairperson

ACKNOWLEDGEMENTS

To Professor Glenn Blank, whose guidance and patience made it possible.

To my parents, whose love made it probable.

To Mona, whose inspiration and encouragement made it happen.

Table of Contents

List of Figures

Abstract

Affix agreement is a common morphological phenomenon in which a grammatical constituent displays an affix that matches the agreement features (person, number, gender, etc.) of another constituent. Register Vector Grammar (RVG) is an architecture for natural language parsing that operates in real-time using fixed finite resources. I propose enhancements to the RVG processor that enable processing of affix agreement within these design constraints. Changes to the lexicon definition will provide the grammar writer with complete flexibility to define the agreement features of the language and the possible values for these features. New run-time data structures will hold agreement vectors of grammatical elements already processed. New syntactic actions test for agreement between these grammatical elements. Several examples are provided to illustrate the operation of these new features and actions. These design enhancements allow for a flexible yet efficient representation of affix agreement processing in the RVG system.

1 INTRODUCTION

Register Vector Grammar is an architecture for natural language parsing first proposed in [Blank 89]. The RVG system comprises a lexicon compiler, a grammar compiler, and a run-time parsing engine. The RVG system allows a grammar writer to create a lexicon source file and a grammar source file for the language or language subset under study, compile these file to RVG internal data format, and run the parser against input sentences using the compiled files. Details of the current RVG system are provided in [Blank 89].

The main theoretical thrust of the RVG parsing system is that natural language can be parsed in real-time using fixed finite resources. Real-time language processing performance is observable in human behavior. The effects of limited short-term memory on such phenomena as center-embedding and processing of ambiguous structures are also observable. Furthermore, many of the structural elements of natural languages form small, closed classes (of pronouns, affixes, grammatical roles, prepositions). These closed classes suggest fixed dimensions of a processing architecture. The design goal of the RVG system is to model these architecture and processing constraints, and at the same achieve real-time processing performance.

A corollary to the principle of fixed, finite resources is that complexities in natural language should not be hidden in an explosion of the grammar size. Components of the processor architecture should work to avoid redundancy in the grammar rules (e.g. for processing a noun phrase in different grammatical roles). The GPSG model, in which metarules and base rules combine to generate huge object grammars, is not acceptable as a performance architecture (the GPSG model makes no commitment to fixed finite resources in performance anyway).

For the same reason, simple finite automata are not acceptable because of the multiplication of states and transitions needed to model phrase embedding and discontinuous constraints in general. RVG complicates the simplicity of finite automata in order to eliminate systematic sources of redundancy. The challenge is to eliminate redundancy without unnecessarily increasing computational complexity.

The major features of the RVG architecture embody the principle of fixed finite resources. The ternary vector state engine is a modified FSA; there is no unbounded stack. Ternary vectors allow propagation of discontinuous grammatical constraints, which reduces grammar size. The boundary backtracking feature allows the state engine to backtrack only to a fixed number of registers, limiting non-determinism. These registers are pre-defined in the grammar and are motivated by salient positions in syntactic structure. See [Blank 89] for details.

In this thesis I describe extensions to the current RVG processor to support affix agreement and grammatical roles. Affix agreement occurs widely throughout the world's natural languages, and can play a role in constraining search during parsing. Grammatical role structures are needed to support affix agreement; they also provide the foundation for semantic interpretation. In the following sections I will review affix agreement and grammatical roles in general terms, then focus on their implementation in the RVG system. I will also pursue a further analysis of the boundary structures of RVG boundary backtracking, and the relationship between boundaries and grammatical roles.

2 AFFIX AGREEMENT

Affix agreement occurs when a language requires that two or more constituents agree for some feature, and at least one constituent is explicitly marked by an inflectional affix. In the word **dogs**, the root form is **dog** and **-s** is an affix indicating PLURAL. Thus in the phrase **the three dogs**, the words **three** and **dogs** agree because **three** as an adjective requires PLURAL and **dogs** indicates PLURAL. In contrast, the phrase **the three dog** is grammatically incorrect.

2.1 Types of Affix Agreement

At first glance, the world's languages display a dazzling confusion of affix agreement phenomena. I will provide just a few examples for us to begin with.

2.1.1 Examples of Affix Agreement

In English, subject-verb agreement occurs for PERSON and NUMBER:

(1)
I see the boat.
He sees the boat.
They see the boat.

Agreement between the verb and the direct object occurs in Spanish when the direct object is definite and human ([Givon 79] p.363):

(2)
le-vi a Juan ayer.
him-saw-I DAT John yesterday.
'I saw John yesterday.'

Lehmann [Lehmann 88] provides an example from Abkhaz of the verb agreeing with its absolutive, ergative, and indirect object actants (Lehmann p.57):

(3)
(sara) a-x c'-k a a-s q '-k a
I ART-child-PL ART-book-PL

0-r -s-to-yt'

ABS.3-DAT.3.PL-ERG.1.SG-give.DYN-FIN

'I give the books to the children.'

In Latin, adjectives display case agreement with the noun (Givon p.375):

	vir	**bonus**	**puero**	**bono**
(4)	**man-NOM**	**good-NOM**	**boy-DAT**	**good-DAT**

librum bonum dedit
book-ACC good-ACC gave

'The good man gave the good boy
a good book.'

The adjective displays definiteness agreement with the noun in Hebrew ([Barlow 88] p.5):

isha tov-a axat
(5) woman good-FEM.SG one-FEM.SG
'a good woman'

ha-isha ha-tov-a
the-woman the-good-FEM.SG
'the good woman'

Finally, in Swahili, gender/class/number agreement occurs throughout the noun phrase and extends even into the relative clause (Givon p.373):

ki-le ki-kapu ch-angu ki-dogo amba-cho ki-me-vunjika
(6) the basket mine small REL it-PERF-break
'that small basket of mine that broke...'

vi-le vi-kapu vy-angu vi-dogo amba-vyo vi-me-vunjika
the baskets mine small REL they-PERF-break
'those small baskets of mine that broke...'

2.1.2 Internal vs. External Agreement

Lehmann divides all agreement phenomena into two classes, which he calls *internal agreement* and *external agreement*. Internal agreement refers to those situations where a determiner or adjective agrees with the NP of which it

is a part. This class also includes some less common phenomena such as agreement of a relative clause with its NP or of a possessor NP with its NP. Examples (4), (5), and (6) above illustrate internal agreement. Lehmann pointedly defines internal agreement as agreement of a constituent with its NP, *not* with the head noun, and provides an interesting example in support of his argument (Lehmann p.58):

```
tri svetlye komnaty
three.NOM.PL light.NOM.PL room.GEN.SG
'three light rooms'
```

He explains:

> In Russian, the lower numerals take their semantic head noun as a genitive attribute in the singular. Nevertheless such an NP is grammatically plural, as becomes evident when it includes an adjective attribute: the adjective...shows nominative plural and thus agrees with its NP, not with its head noun...

Agreement that refers to an NP outside the agreeing term, such as examples (1), (2), and (3), Lehmann calls external agreement. External agreement most commonly occurs between the verb and its NP arguments, but Lehmann provides agreement examples of a possessum with its possessor NP, and of a postposition with its complement (both from the Abkhaz language).

Lehmann concludes that "all agreement refers to an NP" - both internal and external agreement. Internal agreement co-references the agreeing word with its NP; external agreement references an external NP from the agreeing word.

2.1.3 Agreement Hierarchies

Givon and Lehmann both propose hierarchies of case-roles which help to delineate the occurrence of external agreement. Both suggest grammatical role hierarchies of SUBJ > D-OBJ > IND-OBJ. Givon displays an analogous

nominative/accusative case-role hierarchy, while Lehmann provides an ergative/absolutive hierarchy. Givon claims the hierarchy indicates the increasing likelihood of explicit external agreement occurring (for a given language) on the more centrally important roles - those higher on the scale - because of the important place topicality has in agreement phenomena.

Lehmann's stronger claim is that any language which displays external agreement for some role on the hierarchy will also display agreement for all higher roles. He points out the tendency of external agreement to be in complementary distribution with case marking within a language. Each serves an important, but opposite, informational function: agreement specifies a related constituent, leaving the relation between it and the agreeing term implicit, while case marking specifies the relation, leaving the related constituent implicit.

2.2 Eight Points of Inquiry

Barlow suggests eight points of inquiry which must be addressed by any thorough study of agreement. Though I consider this project to be less a study of agreement than the creation of an architecture or environment for the study of agreement, nevertheless the design choices that we make for the RVG agreement system will have some theoretical implications. Because we propose the RVG system as a universal platform for building natural language grammars and parsers, the design choices that we make are implicitly (if not explicitly - as in the 'fixed, finite resources' argument) hypotheses about the universal structure of natural language. If the RVG agreement system is structured such that some hypothetical linguistic activity cannot be modeled in the grammar, then RVG is either admitting of a certain limitation relative to NL parsing, or is pos-

tulating that this linguistic activity is proscribed by some universal constraint on natural languages.

With this in mind, I hope to design an agreement system for RVG which places as few constraints as possible on the grammar writer. The greater the flexibility of the RVG agreement system, the more freedom the grammar writer has to pursue different theories of the nature and structure of agreement phenomena in some specific implementations.

At the same time, this agreement system must be kept small and simple to adhere to the overarching RVG dictum of fixed finite resources. We also do not want to burden the grammar writer with a complex and cumbersome system for creating linguistic models of natural language.

It is in the context of these competing design goals that I would like to highlight Barlow's eight points of inquiry, for they are the dimensions of freedom that the grammar writer may explore. I offer here Barlow's points of inquiry, condensed versions of his definitions of each, and my comments where appropriate. At the conclusion of this paper, I will return to these points and review the implications of the proposed agreement system design with regard to them.

(1) Domain -- what kinds of elements agree with what kinds of elements in what kinds of grammatical configurations

Here we must define the scope of what we consider to be agreement phenomena. Unfortunately, the limits of affix agreement are not always clear. Separating affix agreement from similar or related linguistic phenomena can be difficult.

Two such phenomena that we will not include in this project are semantic agreement and anaphora agreement.

Affix agreement is to be distinguished from semantic agreement (or selectional restriction), which occurs when constituents or their semantic forms must agree for some set of semantic features. For example, the sentence

The carpenter pounded the nails with a hammer

presents no difficulties (other than its triteness), whereas

The carpenter pounded the nails with a sneeze

violates a semantic agreement constraint which expects an INSTRUMENT where it finds the word **sneeze**. The sentence is grammatically correct; however, we are at a loss as to its meaning.

The distinction between affix agreement and semantic agreement will not always be clear because there will often be some overlap between the sets of affix features and semantic features. For example, ANIMATE is an important semantic feature, but it is also used as an affix feature in some languages.

Anaphoric agreement occurs when an anaphoric pronoun agrees with its antecedent, e.g.:

John was hungry, so he ate a second helping.

Here the anaphoric pronoun **he** agrees with its antecedent **John** in PERSON, NUMBER, and GENDER. Some linguists consider this to be part of one phenomenon called grammatical agreement, along with affix agreement. The difficulty is that in order to check the agreement, we must first determine the antecedent of the anaphoric pronoun, which is not at all a trivial problem. Processing anaphoric agreement is beyond the scope of this thesis and will not be discussed further.

(2) Features -- in what properties may grammatical elements agree

These would be the agreement categories such as PERSON, NUMBER, GENDER, CASE, DEFINITENESS, ANIMACY that we see agreement matching occurring for in various languages.

(3) Directionality -- which element is the "controller" and which is the "target", or is agreement non-directional?

Does the verb agree with the subject, the subject agree with the verb, or do they agree with each other?

(4) Strictness -- how exactly do the agreeing sets of categories match up?

Issues that arise here can include partial matching of agreement features, or languages that allow or even require feature mismatches or reversals -- "typified by the Semitic pattern of the numerals '3' to '10' taking the opposite gender of the noun they are in construction with." [Barlow 88]

(5) Conflict -- when two or more patterns of agreement are in conflict, what kinds of "resolution rules" operate?

The most common example of agreement conflict is when two nouns in a subject conjunction have opposite features -- e.g. for gender -- and the language requires verb agreement for that feature. If one noun is MASC and one FEM in a compound subject, should the verb be marked MASC or FEM (or NEUTER?). Different languages may have different rules which provide different answers to this question.

Conflict as defined here is quite different from agreement failure -- when two constituents fail to agree:

*** The boy sing.**

Agreement failure results merely in an ungrammatical sentence. Agreement conflict occurs when one of the constituents, e.g. SUBJECT, provides conflicting agreement information to the other agreement constituent ([Corbett 88] p.43):

```
ta        streha,   okno         in  gnezdo      pod
that.FEM roof.FEM window.NEUT and nest.NEUT under

njim mi     bodo    ostali          vi spominu
it   me.DAT will.PL remain.MASC.PL in memory

'That roof, window, and the nest under it
 will remain in my memory.'
```

Here the compound SUBJECT has one FEM element and two NEUT elements. Following the resolution rules of Slovenian grammar, the verb shows MASC agreement. Corbett (p.43) reports that the Slovenian gender resolution rules are:

```
1) If all conjuncts are feminine, then
      the feminine form is used;
2) otherwise the masculine form is used.
```

The important issue here is that all languages that have agreement most likely have these sorts of resolution rules, because conflict can occur whenever a compound constituent is constructed. The resolution rules themselves may vary widely from language to language.

(6) Variation -- under what circumstances are there alternative agreement options?

In some languages, there will be alternative agreement patterns that are allowed in certain circumstances. These often arise in the conflict situations described above.

Here is an example of a conjoined subject where the elements are not in conflict, yet there is still variation, or choice, in the agreement (Corbett p.25):

```
prepodavalis'  matematika          i   fizika
taught.PL.REFL mathematics.FEM.SG and physics.FEM.SG
'Mathematics and Physics were taught.'

prepodavalas'      matematika          i   fizika
taught.FEM.SG.REFL mathematics.FEM.SG and physics.FEM.SG
'Mathematics and Physics were taught.'
```

The predicate may show either plural or feminine-singular agreement with the conjoined subject.

(7) Function -- what syntactic, semantic, or pragmatic functions may agreement serve?

Various explanations have been put forth of the function that affix agreement serves in language. Barlow offers a brief review of these theories in his introduction (p.17). Some linguists complain of the uselessness or superfluousness of agreement. Some point to its diachronic origins (see below), and suggest that any synchronic functions are secondary. Some see in it an important referential function, identifying an associated noun phrase. Barlow adds his own fanciful suggestion to the list (p.18):

> ...no one seems to have considered the playful, poetic, aesthetic use of language, which is probably of considerable importance in the evolution and acquisition of language as well as contributing to the creative, system-building side of language. It might well be worth exploring the possibility that agreement persists and even spreads in response to the same kind of factors at work in the conventionalization and persistence of rhymed word-pairs, prose rhythms, patterned repititions, and the like.

(8) Change -- what are the diachronic sources of systems of agreement?

Givon has presented the theory that subject-verb agreement evolves from pronouns (p.353):

> Diachronically, independent pronouns may become de-stressed and cliticized, and unstressed/clitic pronouns eventually become agreement inflections on the verb.

Barlow points out that this theory does not apply to all forms of agreement (p.18), and Givon concurs, stating that "the diachronic process via which head-modifier agreement arises has not yet been investigated in any depth." (p.375)

One possibility that the RVG system will allow is that of modeling different stages in the evolution of a language. Any single grammar written in the RVG system provides only a synchronic snapshot of a language. However, one could construct alternative grammars to represent different stages of the language -- e.g. for modeling the evolution of independent pronouns to cliticized pronouns to affix agreement. Thus, Givon's theory could perhaps be given added weight through the construction of concrete grammatical models illustrating the proposed process. In this sense RVG could certainly be used as a tool for modeling theories of diachronic change.

2.3 Requirements for RVG

I hope the examples provided so far in this thesis have convinced you that affix agreement is a phenomenon that occurs in many forms throughout the world's languages. Its very ubiquitousness makes it an unavoidable issue for any project (such as RVG) that aspires to provide general natural language parsing capabilities. Parsers that duck issues of affix agreement are not providing an accurate model of natural language. Affix agreement is a "syntactic glue" holding together various structural elements of a sentence. Sentences sometimes could not be constructed without it; they should not be parsed without it either.

One simple way to model affix agreement is to explode the grammar size. That is, to model SUBJECT/VERB agreement in NUMBER, we create two

SUBJECT-VERB parsing rules for each one we had before: one to parse a SINGULAR SUBJECT and SINGULAR VERB, and another to parse a PLURAL SUBJECT and PLURAL VERB. For agglutinative languages such as Arabic, we would have to multiply the number of rules for each agreement category, because these languages use a separate affix for each agreement feature. That is, to add gender agreement to our example, our two rules would become four: one each for MASC.SG, FEM.SG, MASC.PL, and FEM.PL subject-verb agreement. Clearly this type of combinatorial expansion is unacceptable. Even portmanteau languages would show an undesirable degree of redundancy in parsing rules that tried to handle affix agreement directly.

What we would like to do is implement new structures in the parser which will allow agreement to be handled in a general way. At the same time, however, we must not violate the RVG philosophy of fixed, finite resources.

Too many (in fact, nearly all) existing proposals allow unbounded resources for processing. ATNs permit an arbitrary number of registers, and DCGs allow arbitrary lists of agreement features as rule parameters. I maintain that agreement features can be processed using fixed finite resources and tight computational constraints. Though different natural languages may use different agreement features, for any given language the agreement features make up a small fixed set. Likewise, for each agreement feature the values it can have make up a small fixed set. Thus, for a particular language, the agreement information can be compiled into fixed structures in the grammar and lexicon. There is no need for unbounded resources to process agreement; fixed and finite structures will suffice.

Along with the 'fixed and finite' design goal, I also wish to place as few constraints as possible on the grammar writer, except in those cases where there is theoretical justification for constraints. Again, I will focus on the eight points discussed in the previous section as being the critical issues to address, both in the RVG environment and for the grammar writer.

Finally, I have already mentioned Lehmann's argument that NPs are at the center of all agreement phenomena, and that the NPs most central to the structure of the sentence will be the ones marked for agreement. These NPs correspond to the arguments of the predicate, and to the traditional grammatical roles: SUBJECT, DIRECT OBJECT, and INDIRECT OBJECT. The implication of Lehmann's reasoning is that grammatical roles are central and critical in the phenomenon of affix agreement.

3 GRAMMATICAL ROLES

Grammatical roles are elements of surface syntactic structure that relate each predicate to its arguments. Grammatical roles have names that correspond to traditional grammatical relations, e.g., for English, PRED, SUBJ, OBJ, TOPIC, TENSE, NP, NPMOD. Via these names, actions associated with syntactic productions will have access to information about agreement and pieces of semantic interpretation.

The current RVG system provides only a linear trace of production firings as its completed parse. To implement affix agreement processing, we must enhance the parser to create and maintain grammatical role information. We need to be able to store agreement information in a grammatical role in order to be able to impose constraints on another grammatical role later in the parsing process. For example, if we are parsing the sentence **Those three dogs in the yard chase everyone** we need to save information about the noun phrase **those three dogs** (such as the fact that it is PLURAL) in a grammatical role (in this case, the SUBJECT). This will enable us to check NUMBER agreement when we parse the VERB **chase**.

Notice that the intervening noun phrase **in the yard** may require some affix agreement processing of its own - i.e. agreement processing is not necessarily continuous and linear. We need to be able to impose discontinuous constraints on the input depending on the agreement features. In this case, however, we will handle the discontinuous constraints by using the grammatical role structures, rather than the ternary vector of the syntactic state register (The ternary state vector provides for elegant processing of discontinuous constraints - see [Blank 89] - however, as we discussed earlier, we do not want to

process agreement directly in the state grammar. Therefore we choose to use grammatical role structures instead.)

In most parsers, grammatical roles hold syntactic structures -- parse trees or directed acyclic graphs (DAGs). A complete structural map of the input is constructed and annotated with features. At the opposite extreme, some researchers (especially Schank and his students) prefer to do away with the overhead of syntax altogether and proceed directly to conceptual analysis. The RVG system seeks a middle ground, where a highly efficient syntax, running in small linear time, directly guides semantic analysis. Syntax can provide strong and inexpensive constraints on the potentially vast knowledge-base search needed for direct conceptual analysis.

Unlike most syntactic parsers, RVG does not build syntactic trees (or DAGs) and then transform them into a "deeper" semantic form. An RVG parser maps sentences directly into semantic form. Grammatical roles do not hold syntactic trees decorated by feature structures; rather they are just subscripts into discourse referential structures containing information about semantics and agreement.

4 IMPLEMENTING GRAMMATICAL ROLES IN RVG

The previous section reviewed how grammatical roles are traditionally viewed in parsing systems and how they are viewed in the RVG system (the logical design). This section will elaborate on how grammatical roles fit into the physical design of the RVG data structures.

4.1 Boundaries and Grammatical Roles

Boundary backtracking was proposed in [Blank 89] as a mechanism for providing limited parsing backtracking using fixed finite resources. The current parsing state is saved in a boundary register for possible future backtracking only at pre-defined points in the grammar.

Boundary registers are fixed in number, and saving the current state in a register may cause the loss of information about an earlier backtracking state. Boundary points have been loosely defined as "salient positions in the syntactic structure", and a tentative set of boundaries for English was proposed in [Blank 89]; however, the appropriate set of criteria for determining boundaries has remained an open question. Possible criteria for establishing boundaries may include conjunction and adjunction phenomena. Garden path sentences (such as "The horse raced past the barn fell." or "The prime number few.") provide information about inputs the processor should NOT handle (to mimic human performance); thus backtracking activity and possible boundaries appear to be restricted.

We now propose that the boundaries of boundary backtracking are motivated by and co-incident to the grammatical roles of syntactic structure. Boundaries occur at the points where the RVG parser can identify a grammati-

cal role, and saving a boundary automatically causes the creation of an equivalent grammatical role. Thus, when the parser recognizes the SUBJECT role, it will save the current parsing state in the SUBJECT boundary (for backtracking) along with all the SUBJECT grammatical role information. The SUBJECT grammatical role is saved in the boundary; and when SUBJECT grammatical role information is needed later (for agreement or semantic interpretation) it is accessed through the boundary as well. We will see a more concrete example of this later in this thesis.

4.2 RVG Data Structures

The major data structure used by the RVG parser (aside from the grammar and lexicon) is the set of boundary registers (Figure 1). The boundary registers constitute a named set identified by the grammar writer in the source grammar. Each boundary register is composed of the following sub-structures: the *SynState*, the *References*, and the *Gramrole*.

The SynState is the syntactic state register, itself composed of three levels of ternary state vectors (see [Blank 89] for details of the structure and principles of ternary vectors). The vector at Clauselevel 0 maintains the current parse state of the main clause; the other levels are used to parse embedded clauses.

The References are a priority queue of structures that maintain information about the entities mentioned in the discourse of utterances. Specifically, each Reference contains an agreement vector (explained in the next section) and a pointer to semantic information for RVG semantics [Kreider 90]. The Reference queue contains a fixed number of Reference structures. When no more unused structures are available, the least recently referenced structure is

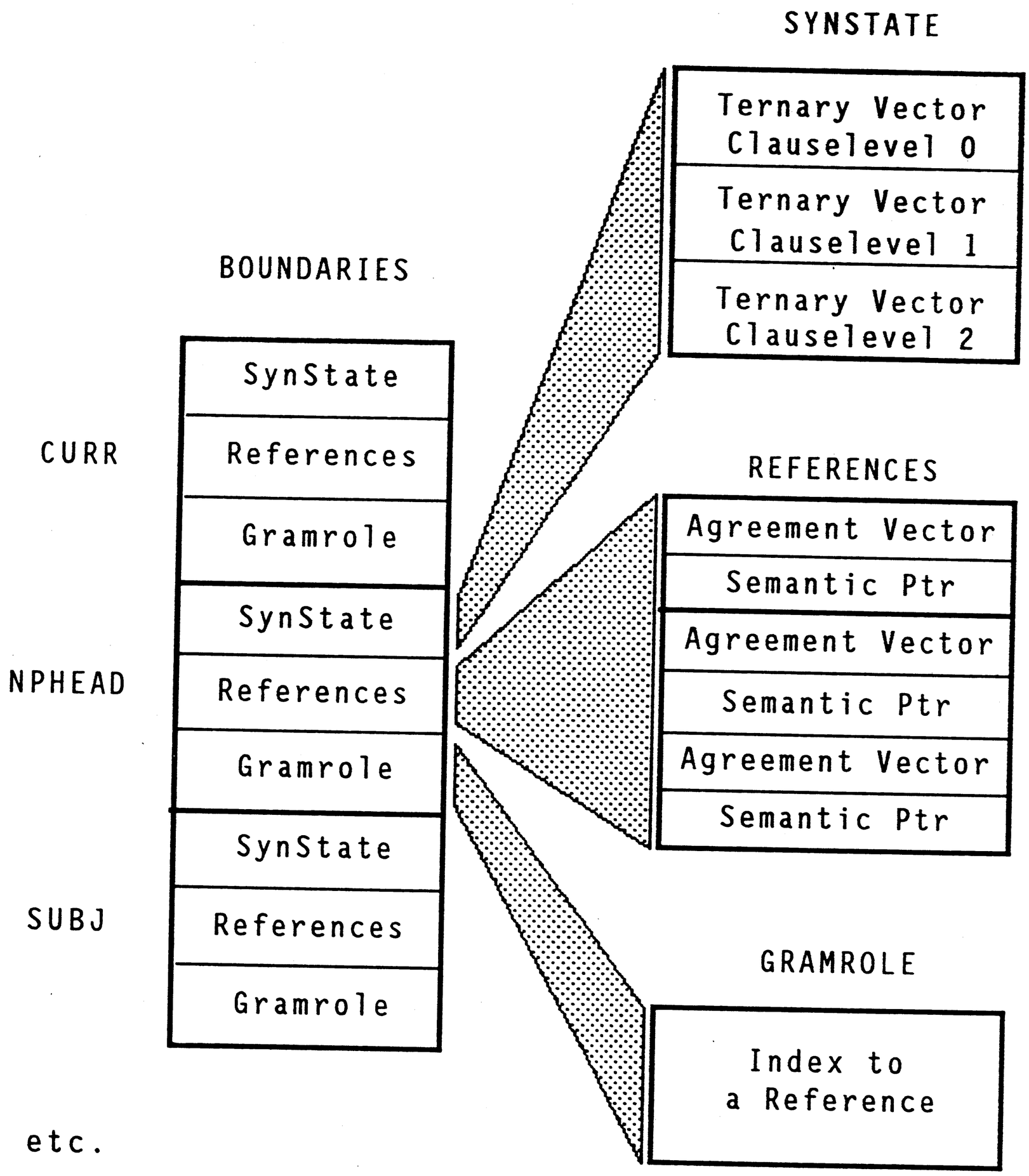

Figure 1: RVG Run-Time Data Structures

"recycled" for a new discourse entity. Thus, old references can eventually be lost (and would then have to be "re-activated" as a new structure if they re-appear in the discourse).

The Gramrole contains an index to a specific Reference. The Reference it indexes is exactly that Reference which describes the discourse entity filling this grammatical role at the current state in the discourse. That is, the Gramrole in the SUBJ boundary will index the Reference containing information about the entity which is the grammatical subject at the current point in the discourse. Thus, the Gramrole provides access to the appropriate Reference for agreement and semantic actions. Note the correspondence of Gramroles with Boundaries: there is exactly one Gramrole for each Boundary.

4.3 Grammatical Role Actions

The data structures design illustrates the close association of grammatical roles with boundaries. Grammatical roles can provide theoretical justification for certain constraints on boundary register activity. Each boundary will be associated with only one position in the left-to-right (for English) processing order, because grammatical roles are associated with a unique entity in the input. The processor will only save a boundary when it has unambiguously recognized a grammatical role -- e.g. only save the SUBJ boundary when it has seen a tense affix and the head of an unmarked NP. These constraints impose greater discipline than [Blank 89], which allowed save actions to occur anywhere in the grammar.

Conversely, grammatical roles are bound only at boundaries. Saving a boundary also binds the corresponding grammatical role to a specific Reference.

There should be no need to change this binding until the boundary is re-used (e.g. the NP boundary may be re-used several times in parsing an input sentence). Therefore, roles are bound only when boundaries are saved, and arbitrary actions to change a role binding are not allowed. Note that the values in the Reference may change - e.g. an agreement action may refine a Reference's affix vector.

We have enhanced the syntax of the actions in the RVG grammar to support this new boundaries/roles design. The *save* action itself is unchanged:

save *boundary*

This action saves the current state (SynState, References, and Gramrole from the CURR boundary) into the named *boundary*. The boundary must be one that was defined by the grammar writer in the **gramroles** section. The *save* action is used to capture a snapshot of the current parsing state for possible later use by the boundary backtracking mechanism.

We have defined a new action for grammatical role assignment with the following syntax:

gramrole1 **:=** *gramrole2*

This action copies the index of *gramrole2* into *gramrole1*, where *gramrole1* can be any of the labels listed in the **gramroles** section and *gramrole2* can be any of these plus CURR. As a result of this action, these two gramroles will now co-index the same Reference. To enforce the design requirement that gramrole binding should be co-incident with boundary saving, the grammar assembler will require that a gramrole assignment action occur in an action list following a save action to the boundary with the same label as *gramrole1*.

Finally, we have defined another new action to set the CURR gramrole to a new Reference:

newref

This action finds an unused Reference (or recycles an old one), clears its agreement vector, assigns it a new semantic memory entry, and sets the CURR gramrole to index that Reference. This action is needed when a new entity will be processed.

Here is an example of how these actions can be used:

```
save SUBJ
SUBJ := NP
newref
```

The first action copies the entire CURR register to the SUBJ register (overwriting any previous values -- this is the boundedness of boundary backtracking). The next action copies the NP gramrole value to the SUBJ gramrole. The SUBJ gramrole now accesses the Reference that the NP gramrole has been indexing. The third action sets the CURR gramrole to a new value so that it is indexing an empty Reference. This allows the parser to begin putting information about a new entity into this Reference.

5 IMPLEMENTING AFFIX AGREEMENT IN RVG

As we discussed earlier, affix agreement occurs on a small number of features, each feature having a small number of possible values. This will allow us to model agreement under the RVG 'fixed, finite' philosophy. We want the parameters of the RVG agreement system bound when the grammar and lexicon source files are assembled, so that at run-time the parser can perform simple efficient operations on fixed finite length data. On the other hand, we want the user interface to be friendly and straight-forward, freeing the grammar writer from mundane low-level data structuring tasks. Let us now examine each of these issues.

5.1 The Affix Agreement Vector

First, the grammar writer must define the set of agreement features for the language, and the possible values for each feature[1]. This will be done in the **morph** section, using braces to delimit the values of one agreement feature. For example:

```
morph  { SNG PL }  { FIRST  SECOND  THIRD }
```

would define agreement values for NUMBER and PERSON in English. When defining this morphological/agreement information, we may find that we have additional morphological features that are not used for agreement. For example, we may want to have a feature indicating GENITIVE. We can add a feature to our **morph** section, but not inside braces:

```
morph  { SNG  PL }  { FIRST  SECOND  THIRD }  GEN
```

The feature values inside braces are mutually exclusive, and are used for agreement operations. Any features outside the braces are independent, and are used

---

[1]I will use the term *agreement features* to mean person, number, and such, and *agreement feature values* to mean singular, plural, etc. In the literature the terminology of *agreement categories* and *agreement features* is sometimes used.

to carry additional morphological information about the lexical entry. These additional features are most commonly used to constrain production firing (explained below).

The RVG assembler will convert the morphological/agreement information that the user has defined into an internal representation suitable for efficient processing. This representation will be a bitvector such that agreement between two lexical entries can be checked merely by intersecting their bitvectors. It is important that we have simple, efficient agreement operations to maintain the RVG design principles. To achieve this, the assembler must define one bit in the internal bitvector for each unique combination of one feature value from each of the feature sets. In addition, one bit will be assigned to each of the additional, "non-agreement" morphological features.

For example, for the **morph** section defined above, the RVG assembler will create the following internal bitvector representation:

```
bit 0:    SNG-FIRST
bit 1:    SNG-SECOND
bit 2:    SNG-THIRD
bit 3:    PL-FIRST
bit 4:    PL-SECOND
bit 5:    PL-THIRD
bit 6:    GEN
```

In this case, bit 0 represents the value SNG of the agreement feature NUMBER, and the value FIRST of the agreement feature PERSON. Each of the first six bits represents one combination of agreement feature values; together the six bits represent all possible combinations. This allows any type of agreement constraint for NUMBER and PERSON to be represented by turning ON or OFF the appropriate bits.

A bitvector representing "singular" would have the bits SNG-FIRST, SNG-SECOND, and SNG-THIRD turned ON, and the other bits turned OFF. A bitvector representing "third-person plural" would have the PL-THIRD bit turned ON, and all the other bits turned OFF. If these two bitvectors were intersected, no bits would remain on, and the agreement would fail. If, however, the second bitvector were intersected with a bitvector for "plural" (PL-FIRST, PL-SECOND, PL-THIRD), then the PL-THIRD bit would remain on as a result and the agreement would succeed.

It might appear that in exchange for efficient run-time processing of agreement, this design creates bitvectors of unwieldy size. In practice, this is quite unlikely to be the case. Most agreement features are binary-valued (MASC/FEM, SNG/PL) or have at most a very small set of values (MASC/FEM/NEUT, SNG/DUAL/PL). In addition, most languages use only a small number of agreement features, so that the agreement vector will in general be quite manageable in size.

We have seen how the grammar writer defines the agreement feature values in the **morph** section which determine the internal agreement bitvector. Now let's examine how she uses the agreement values in the lexicon to define specific agreement bitvectors for each lexical entry. Requiring the user to define bitvectors in terms of the internal representation above (e.g. SNG-FIRST, SNG-SECOND, SNG-THIRD) would be very clumsy and time-consuming; instead we will again let the assembler convert a user representation into our internal bitvector format.

Here are some examples of agreement vectors that the user could enter in the lexicon, with their corresponding internal bitvector representations. A detailed explanation will follow:

- < FIRST > = SNG-FIRST PL-FIRST
  (i.e. all bits with FIRST value are turned ON)
- < SNG > = SNG-FIRST SNG-SECOND SNG-THIRD
  (i.e. all bits with SNG value are turned ON)
- < FIRST SECOND > = SNG-FIRST SNG-SECOND PL-FIRST PL-SECOND
  (i.e. all bits with either FIRST or SECOND value are turned ON)
- < FIRST SNG > = SNG-FIRST SNG-SECOND SNG-THIRD PL-FIRST
  (i.e. all bits with either FIRST or SNG value are turned ON)
- < THIRD*PL > = PL-THIRD
  (i.e. all bits with both THIRD and PL values are turned ON)
- < FIRST*PL THIRD*PL > = PL-FIRST PL-THIRD
  (i.e. all bits with either FIRST and PL, or THIRD and PL are turned ON)

If the user enters "< FIRST >" as the agreement information for a lexical entry, the compiler will create a bitvector with the SNG-FIRST and PL-FIRST bits on. This entry would agree with another entry whose bitvector had either of these bits on. For example, suppose the user defines another lexical entry with "< SNG >" as the agreement vector. This translates to an internal bitvector of SNG-FIRST SNG-SECOND SNG-THIRD. Now note that these two bitvectors agree on the bit SNG-FIRST. This is crucial. The bitvectors with user representations "< FIRST >" and "< SNG >" agree on the bit SNG-FIRST. They agree because <FIRST> constrains the PERSON to "first-person", but does not constrain the NUMBER, while <SNG> constrains the NUMBER to "singular", but does not constrain the PERSON.

The vector that the user defines for a lexical entry indicates *constraints* on agreement; not agreement features. Thus, the empty user agreement vector "< >" imposes no constraints, and is assembled into the internal bitvector with all agreement bits on (the setting of the additional morph bits will be explained later). Finally, note that "< >" is equivalent to <SNG PL> and to <FIRST SECOND THIRD>.

My hope, of course, is that the user need not think about the internal bitvector representation at all. Having defined the set of agreement features in the **morph** section, she can then write the user agreement vectors in the lexicon directly from the nature of the lexical entries, without thought to the machine implementation of agreement processing.

5.2 Additional Morphological Information

Additional morphological information can be included in the internal bitvector by defining other morphological symbols outside the agreement feature sets in the **morph** section (as was shown with the GEN symbol to indicate "genitive"). Each of these symbols will indicate one binary piece of morphological information, and will map to one bit in the internal bitvector.

When the RVG assembler reads the **morph** section, it will create a bitvector mask corresponding to just the agreement bits in the internal bitvector. This mask will be used when agreement checks occur (i.e. bitvectors are intersected) to ensure that only the agreement bits are checked, and not the additional morphological features.

The additional morphological features can be used for two purposes: to constrain production firing, and to carry information for semantic or pragmatic analysis. At this time, we will only be concerned with how the morphological features can constrain production firing. When a production is defined in the RVG grammar, a morphological vector can be defined for that production. This morphological vector may contain any of the bits that have been defined as additional morphological features in the **morph** section. The production's morphological vector constrains the production such that the production may only fire if its morphological vector "agrees" with the morphological features of the current lexical entry. That is, the intersection of the production morphological vector and the additional morphological features of the lexical entry must be non-empty.

The agreement features and the non-agreement features in the **morph** vector are completely separate; there is no interaction between them. The agreement features are used in agreement operations, and the non-agreement features are used to constrain productions. They could be defined in two separate vectors (and might well be better off that way). I have only chosen to leave them together because the current structure of the RVG system supports one morph vector, and changing that would require extensive revisions throughout the system.

Now that we have described the structure of the morph vector from both the user and the internal perspectives, let's examine one more example. Here is a simple paradigm as it might be defined in the lexicon:

```
morph { SNG PL }  { FIRST SECOND THIRD }  GEN

paradigms
```

```
m BED
   /    < SNG >
   s    < PL >
   's   < SNG GEN >
   s'   < PL GEN >
```

Each paradigm defines a unique set of affixes and their corresponding agreement vectors. This set of affixes may be valid for many lexical entries. When an individual lexical entry is defined, it can refer to the appropriate paradigm to indicate the set of affixes it will take.

In this example, a paradigm BED is defined. The / symbol indicates a zero-affix, and its agreement vector is <SNG>. The **s** affix is defined with an agreement vector of <PL>, and so on. Thus, the definition of affixes and agreement vectors is condensed into one instance, instead of being duplicated for each lexical entry with this affix pattern.

5.3 Affix Agreement Actions

Besides the agreement vectors in the lexicon and the productions, the grammar writer must add one other thing to the RVG input files to complete the agreement system: the 'agree' action itself must be added to the grammar in the appropriate places. A new syntactic action, 'agree', will be added to the RVG system to be used for agreement checking. It may have one of three forms:

(a) *gramrole* **agree** *gramrole*

(b) *gramrole* **agree lex**

(c) *gramrole* **agree** *agree_spec*

In all cases a *gramrole* may be either a user-defined grammatical role or the CURR role.

An **agree** action in notation (a) above means that the agreement vectors found in the References corresponding to each *gramrole* are bit-intersected (masked for only the agreement bits) to test the agreement. If there is a bit on in the result vector, then the agreement succeeds. If so, the result vector is loaded into the agreement vector of the lefthand *gramrole*, and parsing continues. If the agreement fails, then the production fails and the parser must try another production, or backtrack.

In notation (b), the agreement vector from the *gramrole*'s Reference is tested for agreement with the agreement vector returned by lexical lookup for the current lexical entry. Again, if agreement succeeds the result is loaded into the *gramrole*'s agreement vector. In practice, this action may often be used as

```
CURR agree lex
```

to load the lexical entry's agreement vector into CURR. If the CURR gramrole has just been re-initialized to an empty Reference (by the **newref** action) then the **agree** action in effect just loads the lexical agreement vector into CURR's Reference. A newly initialized Reference has an agreement vector with all bits on, so the bit intersection operation just loads in the other vector.

Finally, the grammar writer may use **agree** to test a gramrole's agreement vector against an explicit specification, such as:

```
SUBJ agree <THIRD*SG>
```

to test the agreement vector of the SUBJ role's Reference for agreement with the third person singular.

Each form of the *agree* action can be used to perform a slightly different task. The first variant actually enforces agreement between grammatical roles.

The second variant retrieves affix agreement material from the lexicon. The third variant allows the grammar to specify agreement constraints above and beyond those imposed by individual lexical entries.

5.4 Affix Agreement Examples

I will now demonstrate how this affix agreement design works by proceeding step-by-step through some simple examples. The lexicon and grammar fragments that I provide for these examples are for illustrative purposes only; I do not mean to imply that they are appropriate RVG representations of a comprehensive English grammar. The first examples will illustrate the processing of subject-verb and determiner-head agreement in English.

5.4.1 Internal and External Agreement in English

Let us continue to use the **morph** vector we have already defined:

```
morph {SG PL} {FIRST SECOND THIRD}
```

Thus the internal bitvector for this grammar (always shown inside double angle brackets) is:

```
<<FIRST-SG SECOND-SG THIRD-SG
  FIRST-PL SECOND-PL THIRD-PL>>
```

The lexicon we will use is shown in Figure 2.

Now let's see what happens when the parser is given the input **A clock ticks**.

(1) The production NP fires, and executes the actions **save NP** and **newref**. The first action saves the CURR register values to the NP boundary, and the second action assigns a new reference to CURR. This creates two accessible references, to allow for the possibility of left-embedded genitives ("A neighbor's clocks...").

```
paradigms

m BED
   /              <THIRD*SG>
   s              <THIRD*PL>

m SHEEP
   /              <THIRD>

m PULL
   /              <FIRST SECOND THIRD*PL>
   s              <THIRD*SG>

m A
   /              <THIRD*SG>

m THE
   /              <THIRD>


entries

e clock        cat NOUN          morph clock_BED_
e sheep        cat NOUN          morph sheep_SHEEP_
e tick         cat VINTRANS      morph tick_PULL_
e graze        cat VINTRANS      morph graze_PULL_
e a            cat DET           morph a_A_
e the          cat DET           morph the_THE_
```

Figure 2: Sample RVG Lexicon

(2) The production DET fires, and executes the action **CURR agree lex**. The lexical entry is **A**, which has an agreement vector of **<<THIRD-SG>>**. The CURR role takes this vector.

(3) The production HEAD fires, and executes the actions **NP agree CURR** and **NP agree lex**. The first action merges material in the CURR agreement vector (including the agreement vector from the determiner) with any material in the NP register. The second action tests for agreement between the current lexical entry, **clock**, and the NP register. Since the entry **clock** has an agreement vector of **<<THIRD-SG>>**, the agreement between the determiner and the head succeeds. Finally, the action **newref** executes to assign a new reference to CURR.

(4) The production TENSE fires, and executes the action **CURR agree lex**. The lexical entry is **ticks**, which has an agreement vector of **<<THIRD-SG>>**. The CURR role (new Reference) takes this vector. Next the actions **save TENSE** and **newref** are executed, to save the state again and copy the gramrole value from CURR to TENSE. The TENSE gramrole now references the agreement vector from the tensed verb.

(5) The production SUBJ now fires because we have seen the tensed verb and the unmarked NP. This production executes the actions **save SUBJ** and **SUBJ := NP**, which save the state from CURR to SUBJ and copy the gramrole value from NP to SUBJ. We have now made **A clock** the subject of the sentence. Finally, the action **SUBJ agree TENSE** is executed to check the subject-verb agreement. The agreement vectors from the References indexed by the SUBJ and TENSE gramroles are intersected, and the result (**<<THIRD-SG>>**)

is stored in the SUBJ agreement vector. The agreement has succeeded so parsing continues.

Now, for comparison, let's see what happens with the input **A clock tick**. All parsing activity for steps (1), (2) and (3) is identical.

(4) The production TENSE fires, and executes the action **CURR agree lex**. The lexical entry is **tick**, which has an agreement vector of **<<FIRST-SG SECOND-SG FIRST-PL SECOND-PL THIRD-PL>>**. The CURR role (new Reference) takes this vector. The actions **save TENSE** and **newref** execute also, so that the TENSE gramrole is indexing the agreement vector listed just above.

(5) The production SUBJ fires as before, executing the actions **save SUBJ** and **SUBJ := NP**. Finally, the action **SUBJ agree TENSE** is executed to check the subject-verb agreement. The SUBJ vector is **<<THIRD-SG>>**, and so the two vectors have no bits in common. The agreement fails, the parser is unable to find an alternative parse, and the input sentence is rejected.

Now let's look at another example: **The sheep graze**.

(1) The production NP fires, and executes the actions **save NP** and **newref**.

(2) The production DET fires, and executes the action **CURR agree lex**. The lexical entry is **The**, which has an agreement vector of **<<THIRD-SG THIRD-PL>>**. The CURR role takes this vector.

(3) The production HEAD fires, and executes the actions **NP agree CURR** and **NP agree lex**. The lexical entry is **sheep**, which has an agreement vector of **<<THIRD-SG THIRD-PL>>**. The determiner/head agreement succeeds, and the action **newref** is executed. The NP gramrole now indexes the Reference for the noun phrase. The agreement vector is **<<THIRD-SG THIRD-PL>>**, which means that the phrase could be singular or plural.

(4) The production TENSE fires, and executes the action **CURR agree lex**. The lexical entry is **graze**, which has an agreement vector of **<<FIRST-SG SECOND-SG FIRST-PL SECOND-PL THIRD-PL>>**. The CURR role (new Reference) takes this vector. The actions **save TENSE** and **newref** execute also, so that the TENSE gramrole is indexing the agreement vector listed just above.

(4) The production SUBJ fires, because the prerequisite tensed verb and unmarked NP have been parsed. The actions **save SUBJ** and **SUBJ := NP** are executed, making the noun phrase the subject of the sentence. Finally, the action **SUBJ agree TENSE** is executed to check the subject-verb agreement. The two vectors agree with a result of **<<THIRD-PL>>**. The result is stored in the SUBJ vector, so that the SUBJ has now been constrained to the plural by the verb.

5.4.2 Inversion Agreement

Now let's look at an example of subject-verb inversion in English. How will RVG process the input **Does the clock tick?**

(1) The production QUES fires, and executes these actions:

```
CURR agree lex
save TENSE
newref
```

to save the state in TENSE and set the gramrole pointing to the agreement vector **<<THIRD-SG>>**.

(2) The production NP fires, and executes the actions

```
save NP
newref
```

(3) The production DET fires, and executes **CURR agree lex**, which loads **<<THIRD-SG THIRD-PL>>** into CURR's gramrole.

(4) The production HEAD fires, and executes these actions:

```
NP agree CURR
NP agree lex
newref
```

The CURR agreement vector is merged into the NP role and refined to **<<THIRD-SG>>** (**clock** must be singular).

(5) The production SUBJ fires, as we have seen the tensed verb and the unmarked NP. The actions

```
save SUBJ
SUBJ := NP
SUBJ agree TENSE
```

are executed. The SUBJ role indexes the NP agreement vector, and the **agree** action succeeds because both roles have **<<THIRD-SG>>** vectors. The parser continues with the input.

5.4.3 External Agreement in Abkhaz

I will now attempt to sketch out how RVG agreement processing would work for one example each of internal and external agreement in other languages. First let's look at example (3) from section 2:

```
(sara) a-x  c'-k a      a-s q '-k a
I      ART-child-PL     ART-book-PL

          0-r -s-to-yt'
ABS.3-DAT.3.PL-ERG.1.SG-give.DYN-FIN

'I give the books to the children.'
```

In this example, the predicate **give** shows affix agreement for person with the absolutive argument, and for person and number with the dative and ergative arguments. The agreement vector definition is:

```
morph { ERG ABS DAT } { 1ST 2ND 3RD } { SNG PL }
```

The case role features will be used to allow the predicate to test for person and number agreement with each of the three noun phrases. In the lexicon, the noun affixes will have agreement vectors that specify person and number, e.g. <3RD*PL> for the affix shown for the word **book**. Since no case feature is specified, this vector will agree with any case; the noun inherently has no case, and could be used in any case role.

The predicate affixes in the lexicon will specify person, number, and case, e.g. <DAT*3RD*PL ABS ERG> for the dative affix of the predicate. The ABS and ERG features are listed to allow any features for those cases to "pass through". Lexical lookup builds the agreement vector for the lexical entry by combining (bit intersection) the internal bitvectors of all the affixes on the word. In this example, there are three affixes with agreement vectors of

```
<ABS*3RD DAT ERG>
<DAT*3RD*PL ABS ERG>
<ERG*1ST*SNG ABS DAT>
```

Each vector specifies the exact features for the case role being matched, but allows all features for the other two case roles. When the vectors are combined by lexical lookup, the final agreement vector for this lexical entry is:

```
<ABS*3RD DAT*3RD*PL ERG*1ST*SNG>
```

which is exactly what we want.

We also need to define three grammatical roles to hold the noun phrases as we parse: ERG, ABS, and DAT. As each noun phrase is parsed, actions are executed to save the agreement vector to the corresponding role, and then to refine it to be specific to that role. For example, when the dative NP is parsed, the actions:

```
CURR agree lex
save DAT
DAT agree <DAT>
```

are executed. The first two actions move the agreement vector <3RD*PL> to the DAT role. The *agree* action refines this vector to be <DAT*3RD*PL> (i.e. it will no longer agree with the ABS or ERG cases). Each of the noun phrases is parsed in this manner. Then when we parse the predicate, the actions

```
CURR agree lex
save TENSE
ABS agree TENSE
ERG agree TENSE
DAT agree TENSE
```

are executed to check the agreement between the predicate and each of the case roles. The dative role agrees because both vectors have the DAT-3RD-PL bit on. The other roles agree as well, and the parser continues.

5.4.4 Internal Agreement in Latin

Next let's look at example (4) from section 2 for processing internal agreement:

```
vir       bonus    puero     bono
man-NOM good-NOM boy-DAT good-DAT

librum     bonum  dedit
book-ACC good-ACC gave

'The good man gave the good boy
 a good book.'
```

For this example our agreement vector will need to represent the case feature, with values of { NOM DAT ACC }.

For each noun phrase in this input, the following sequence of events will occur:

(1) The production NOUN fires, and executes these actions:

```
CURR agree lex
save NP
newref
```

These actions will save the state in the NP boundary and load the agreement vector index into the NP role.

(2) The production ADJ fires and executes the action **NP agree lex**. This action checks case agreement between the head noun and the adjective that follows it.

6 PROGRAMMING DETAILS

In this section I provide a descriptive specification of the programming changes that are needed to implement the proposed affix agreement system in the RVG system, by modifying the most current version at this date (3.5), implemented in Objective-C and YACC on Sun workstations.

6.1 RVG Object Classes

6.1.1 New Object Class Definitions

Two new object class definitions are needed to implement the proposed agreement system. I list them below with a description of the structure and functionality required for each.

(1) RefQueue -- The RefQueue object class will maintain the priority queue of References. It should have the following private variables:

- a fixed length array of References
- an equivalent array of time-stamp values to record the last access time for each Reference (used when it needs to find a Reference to re-use)

Methods must be provided to:

- return an index to a new Reference (may require "recycling" an old Reference)
- set or get an agreement vector given an index to a Reference (using the corresponding Reference methods)

(2) Reference -- This class represents the references. At this time it doesn't do too much, so it could perhaps be implemented as a standard data structure instead. Private variables are:

- an agreement bitvector
- a pointer to semantic information

Methods must be provided to set and get the agreement vector.

6.1.2 Object Class Modifications

The existing RVG object class StateReg needs to be modified to implement affix agreement. StateReg represents an RVG state register. The BRegs structure is an array of StateRegs, each of which is associated with a syntactic boundary. The following data fields need to be added to StateReg:

- String BoundaryLabel
- id RefQueue
- int Gramrole

Additional methods must be provided to:

- initialize all the fields to an appropriate state for beginning a sentence
- set and get the BoundaryLabel name
- set and get the Gramrole index value
- set and get the agreement vector (using the corresponding RefQueue methods and passing the Gramrole value)
- store a boundary, i.e. set SynState, RefQueues and Gramrole values
- resume a boundary, i.e. get SynState, RefQueues and Gramrole values when backtracking

6.2 Grammar Assembler Modifications

The specification of the grammar source file has changed to incorporate the features needed for the affix agreement system. The new specification is shown in Appendix A along with a description of the changes. I am indebted to Carmel Owens for providing the original grammar and lexicon source file specifications in [Owens 90].

6.3 Lexicon Assembler Modifications

The specification of the lexicon source file has likewise changed and is shown along with its description in Appendix B.

Note that the **paradigms** section of the lexicon source file is not shown in this specification because it is not parsed by the lexicon assembler but rather by a separate program that builds the lexical lookup trie. The paradigms can have an agreement vector specified for each distinct affix listed in the paradigm. The agreement vector formerly had the following specification:

```
agreevec   ::= LBRACKET agreelist RBRACKET

agreelist  ::= agreelist NAME
           ::= NAME
```

Under the proposed agreement system, the new specification for this agreement vector will be:

```
agreevec   ::= LBRACKET agreelist RBRACKET

agreelist  ::= agreelist agreeterm
           ::= /\

agreeterm  ::= agreeterm ASTERISK NAME
           ::= agreeterm NAME
           ::= NAME
```

As the RVG assembler parses the agreement vectors in the **paradigms** section, it will need to convert each agreement vector into its internal bitvector representation to be stored in the lexical lookup trie.

One way to do this is to have the assembler first create the internal bitvector corresponding to each individual feature value defined in the **morph** section. For example, given a **morph** definition of:

```
morph { SG PL } { 1ST 2ND 3RD }
```

the full internal bitvector would be:

```
<< SG-1ST SG-2ND SG-3RD PL-1ST PL-2ND PL-3RD >>
```

and the assembler would create an internal data table like this:

Feature Value	Internal Bitvector
SG	<< SG-1ST SG-2ND SG-3RD >>
PL	<< PL-1ST PL-2ND PL-3RD >>
1ST	<< SG-1ST PL-1ST >>
2ND	<< SG-2ND PL-2ND >>
3RD	<< SG-3RD PL-3RD >>

Then, as the assembler parses the lexical agreement vectors, it need only combine these internal representations in simple bit operations to create the correct final internal bitvector for that lexical affix. If two feature values are listed in the lexical agreement vector, e.g. < A B >, then the assember retrieves the bitvectors corresponding to each value and performs a bitwise OR on them to create the proper result. If two values are listed with '*', e.g. < A * B >, then the bitvectors are ANDed together to create the result. Any lexical agreement vector can be properly converted to its internal representation simply by applying these rules and using the precedence of AND before OR, e.g. < A B * C > is processed as:

```
bitvector of A OR (bitvector of B AND bitvector of C)
```

The following algorithm will generate the internal bitvector for each feature value:

```
Let F = number of features
Let V[i] = number of values for feature i
Let N = number of bits in bitvector
          (i.e.  V[0] * V[1] * ... * V[F-1] )
Let B[i,j,k] = bit k of bitvector for value j
                of feature i

n := N
FOR i := 0 TO F-1
    p := n / V[i]
    FOR j := 0 TO V[i]-1
        FOR k := 0 TO N-1
            IF (k mod n) / p = j
            THEN B[i,j,k] := 1
n := n / V[i]
```

6.4 Parser Modifications

The most significant changes that are needed for the run-time system are the new functions to implement the actions. They are as follows:

(1) gramrole assignment

gramrole1 **:=** *gramrole2*

PSEUDOCODE:

```
IF gramrole2 = UnboundValue
THEN error
ELSE
    set gramrole1 index to gramrole2 index
```

(2) set CURR gramrole to new reference

newref

PSEUDOCODE:

```
get next Reference index using RefQueue
    object method
clear Reference's agreement vector
get new SemPtr for Reference
set CURR gramrole to index Reference
```

(3) agreement action

(a) *gramrole1* **agree** *gramrole2*

```
PSEUDOCODE:

    IF gramrole1 = UnboundValue THEN error
    ELSE IF gramrole2 = UnboundValue THEN error
    ELSE
        get gramrole1 agreement vector
        get gramrole2 agreement vector
        result := agreevec1 BIT-AND agreevec2
        IF result = 0 THEN return(FAILURE)
        ELSE
            set gramrole1 agreement vector to
                result
            return(SUCCESS)
```

(b) *gramrole* **agree lex**

```
PSEUDOCODE:

    IF gramrole = UnboundValue THEN error
    ELSE
        get gramrole agreement vector
        get lex entry agreement vector
        result := agreevec1 BIT-AND agreevec2
        IF result = 0 THEN return(FAILURE)
        ELSE
            set gramrole1 agreement vector to
                result
            return(SUCCESS)
```

(c) *gramrole* **agree** *agreement_vector*

```
PSEUDOCODE:

    IF gramrole = UnboundValue THEN error
    ELSE
        get gramrole agreement vector
        result := agreevec1 BIT-AND agreevec2
        IF result = 0 THEN return(FAILURE)
        ELSE
            set gramrole1 agreement vector to
                result
            return(SUCCESS)
```

7 CONCLUSIONS

In this thesis I have proposed an architecture for affix agreement processing in the Register Vector Grammar parsing system. I have strived to make the agreement system as flexible as possible, while still adhering to the RVG principles of real-time processing with fixed finite resources. I would now like to return to Barlow's eight points of inquiry, and review the proposed agreement processing design in light of them.

7.1 Eight Points of Inquiry - Review

(1) Domain -- what kinds of elements agree with what kinds of elements in what kinds of grammatical configurations

The proposed system is quite flexible in this area. The grammar writer may provide an agreement vector with any lexical entry. She may define any number of gramroles to hold lexical agreement information. She may write *agree* actions for any pairs of gramroles in any productions. There are no *a priori* constraints imposed by the agreement system on which constituents may participate in an agreement action.

(2) Features -- in what properties may grammatical elements agree

Here again the proposed system is very flexible. The grammar writer has complete control over the definition of agreement features and the possible values for these features.

(3) Directionality -- which element is the "controller" and which is the "target", or is agreement non-directional?

The RVG *agree* action is directional: the lefthand gramrole is updated with the

result of the agreement bitvector intersection. The grammar writer does have control over what gramrole appears on the left and what gramrole appears on the right.

The grammar writer may also simulate non-directional agreement by using two *agree* actions with the gramrole positions switched, e.g.

```
SUBJ agree TENSE
TENSE agree SUBJ
```

This will place the agreement result into both gramroles. Thus, the system can be made somewhat flexible on this point despite its bias toward directionality.

(4) Strictness -- how exactly do the agreeing sets of categories match up?

Here we arrive at the "irregular" cases. The proposed system may not be expressive enough to deal with these difficult situations elegantly. It is not easy to predict how flexible the system will be for these cases without studying specific examples in detail.

One that we can examine is Barlow's example of the Semitic numerals '3' to '10' taking the opposite gender of their head noun. One approach to this problem would be to make a separate paradigm for these numerals in the lexicon, with the gender of the affixes backwards -- i.e. mark the feminine affix as MASC, and vice versa. Then the agreement action would succeed.

If this solution is not satisfactory (perhaps because we want the true gender meaning of the affix shown in its agreement vector), then we would need to get involved with special syntactic productions for these numerals -- one for each gender. If the numeral had a feminine affix, it would only allow a certain

production to fire that had an action of **NP agree <MASC>**. There would be an analogous production for the masculine affix.

At any rate, I believe that in general it will be possible to construct grammars that can process these special cases, although they may not always do so in what would be considered a straightforward and elegant fashion. The question is whether it is unjustified to have to construct "irregular" grammar productions to process these "irregular" agreement cases.

The alternative is to add much more power to the syntactic actions, such as by providing conditional control. This would allow agreement actions to be executed conditionally, perhaps depending on certain agreement features of the current lexical entry. Adding capabilities such as these will of course bring along entirely new sets of problems to address.

(5) Conflict -- when two or more patterns of agreement are in conflict, what kinds of "resolution rules" operate?

Here again we immediately find ourselves wishing for some control structures in the syntactic actions. The resolution rules that are invoked when conflict occurs may not result in simple testing of agreement features. They may require an active rule to insert the correct feature value into the test -- e.g. the example shown earlier where the verb shows a MASC affix when the conjoined subject has FEM and NEUT affixes. It is very difficult to say whether these phenomena can be adequately represented within the proposed design; certainly it will be a bit cumbersome.

(6) Variation -- under what circumstances are there alternative agreement options?

The example shown earlier for variation again has a conjoined subject, just as the example of conflict does. The flexibility of the agreement system to handle these phenomena of conflict and variation will depend to some extent on how conjunction is handled in the processor. If special syntactic productions are needed to open and close conjoined subjects, and perhaps even special gramroles to process the semantics, then the agreement system may be able to take advantage of these to handle the unusual agreement patterns that occur.

Otherwise, the agreement system itself may need to force changes in the syntactic processing (i.e. additional productions or gramroles) to handle these difficult cases. This is an area where no answer will be clear until attempts are actually made to build RVG grammars for these languages.

(7) Function -- what syntactic, semantic, or pragmatic functions may agreement serve?

The function of agreement is essentially a theoretical issue, and I do not see any way in which the proposed agreement system constrains theories of agreement function.

(8) Change -- what are the diachronic sources of systems of agreement?

This topic is also theoretical in nature. As I discussed earlier, the grammar writer will be able to model grammars at different stages of diachronic change. I do not see any constraints on diachronic theory imposed by the proposed system.

7.2 Future Directions

Two unresolved issues present themselves for future work in this area. The first is how we should model conjoined noun phrases in RVG. It is common

for the conjoined NP to take on feature values that are distinct from the values of its constituents. Will we maintain a separate gramrole to represent the conjunction of the individual constituents? Will it be not a gramrole but a Reference? How will the appropriate features be established for the conjoined NP?

I have not explored these issues deeply enough to have any answers ready for these questions; to my knowledge neither has anyone else.

The second question is that of whether we really need control structures in the syntactic actions of RVG. This question is related to the first one, because conjoined NPs are one of the more difficult things to model in the existing/proposed system. If it is not possible to model them within the data structures I have proposed, then we may need some control structures to make this possible.

The question of control structures is probably the most serious design question facing the proposed system. I have chosen for the time being to stay with a small, simple system that is straightforward to implement and to use. The system I have proposed will allow the grammar writer to model most cases of affix agreement. If at some time we decide this system is inadequate, we will be able to build upon this foundation to implement the enhancements that are desired.

REFERENCES

[Barlow 88] Barlow, M. and C. Ferguson, eds. *Agreement in Natural Language*. Menlo Park, CA: Center for the Study of Language and Information, 1988.

[Blank 89] Blank, G. "A Finite and Real-time Processor for Natural Language". In *Communications of the ACM*, Vol 32, No. 10, 1989, pp. 1174-1189.

[Corbett 88] Corbett, G. "Agreement: A Partial Specification Based on Slavonic Data", in *Agreement in Natural Language*. Menlo Park, CA: Center for the Study of Language and Information, 1988, pp. 23-53.

[Givon 79] Givon, T. *Discourse and Syntax, Syntax and Semantics*, Volume 12. New York, NY: New York Academic Press, 1979.

[Kreider 90] Kreider, K. "Semantic Ambiguities as Vagueness in Register Vector Grammar", Master's Thesis, Lehigh University, in press.

[Lehmann 88]Lehmann, C. "On the Function of Agreement", in *Agreement in Natural Language*. Menlo Park, CA: Center for the Study of Language and Information, 1988, pp. 55-65.

Appendix A
Grammar Specification

A.1 Grammar Specification Changes

The following grammar specification, along with the syntactic and semantic description, is from [Owens 90]. I have updated it to include the changes needed for affix agreement processing. Rules 1, 3, and 25 have been changed. Rules 30 to 34 have been added. Rules 8 and 9 are obsolete.

```
1     rvg_syntax          ::=  mprop_section feat_section
                               default_section bound_section
                               macro_section prod_section
                               sem_section quant_section

2.1   mprop_section       ::=  MORPHS mprop_list
2.2                       ::=  /\

3.1   mprop_list          ::=  morph_term
3.2                       ::=  mprop_list morph_term

4     feat_section        ::=  FEATURES feat_list

5.1   feat_list           ::=  NAME
5.2                       ::=  feat_list NAME

6.1   bound_section       ::=  BOUNDARIES boundary
6.2                       ::=  /\

7.1   boundary            ::=  NAME
7.2                       ::=  boundary NAME

[DELETE RULE 8]
8.1   grole_section       ::=  GRAMROLES grole_list
8.2                       ::=  /\

[DELETE RULE 9]
9.1   grole_list          ::=  NAME
9.2                       ::=  grole_list NAME

10.1  sem_section         ::=  SEMROLES sem_list
10.2                      ::=  /\

11.1  sem_list            ::=  NAME
```

```
11.2                    ::=  sem_list NAME

12.1 quant_section      ::=  QUANTS quant_list
12.2                    ::=  /\

13.1 quant_lists        ::=  NAME
13.2                    ::=  quant_list NAME

14.1 default_section    ::=  DEFAULT COND feat_vector
14.2                    ::=  /\

15.1 macro_section      ::=  MACROS macro_type
15.2                    ::=  /\

16.1 macro_type         ::=  FEAT macro_list

17.1 macro_list         ::=  macro_entry
17.2                    ::=  macro_list macro_entry

18   macro_entry        ::=  DEFMACRO macro_vector

19.1 macro_vector       ::=  PLUSFEAT check_range
19.2                    ::=  MINUSFEAT check_range
19.3                    ::=  QUESFEAT check_range
19.4                    ::=  MACRO
19.5                    ::=  macro_vector PLUSFEAT check_range
19.6                    ::=  macro_vector MINUSFEAT check_range
19.7                    ::=  macro_vector QUESFEAT check_range
19.8                    ::=  macro_vector MACRO
19.9                    ::=  /\

20   prod_section       ::=  PRODUCTIONS prod_list

21.1 prod_list          ::=  production
21.2                    ::=  prod_list production

22   production         ::=  PWORD NAME NAME morph_props COND
                             feat_vector CHANGE feat_vector
                             actions

23.1 actions            ::=  ACTION action_list
23.2                    ::=  /\

24.1 action_list        ::=  action
24.2                    ::=  action_list action

25.1 action             ::=  SAVE NAME
25.2                    ::=  NAME ASSIGN NAME
25.3                    ::=  NAME AGREE agree_vec
```

```
25.4                    ::=  NAME AGREE NAME
25.5                    ::=  NAME

26.1 morph_props        ::=  MPROP morph_prop
26.2                    ::=  /\

27.1 morph_prop         ::=  NAME
27.2                    ::=  morph_prop NAME

28.1 feat_vector        ::=  PLUSFEAT check_range
28.2                    ::=  MINUSFEAT check_range
28.3                    ::=  QUESFEAT check_range
28.4                    ::=  MACRO
28.5                    ::=  feat_vector PLUSFEAT check_range
28.6                    ::=  feat_vector MINUSFEAT check_range
28.7                    ::=  feat_vector QUESFEAT check_range
28.8                    ::=  feat_vector MACRO
28.9                    ::=  /\

29.1 check_range        ::=  RANGE
29.2                    ::= /\

30.1 morph_term         ::=  LBRACE agree_values RBRACE
30.2                    ::=  NAME

31.1 agree_values       ::=  NAME
31.2                    ::=  agree_values NAME

32.1 agree_vec          ::=  LBRACKET agreelist RBRACKET

33.1 agreelist          ::=  agreeterm
33.2                    ::=  agreelist agreeterm

34.1 agreeterm          ::=  agreeterm NAME
34.2                    ::=  agreeterm ASTERISK NAME
34.3                    ::=  NAME
```

A.2 Grammar Syntax

All labels used in the grammar are strings of combinations of letters, digits, and underscores, starting with a letter.

1 This is the outline of the whole grammar specification. A grammar specification consists of the following sections: morphosyntactic properties section, feature section, default vector section, boundary register section, macro section, production section, semrole section, quant section, and actions section.

2 & 3 The morphosyntactic properties section is optional. It begins with the word morphosyntactic_properties and is followed by a list of morphosyntactic property labels and agreement feature value sets.

4 & 5 The feature section is required. It gives the labels for the features used in the grammar. These features represent the sequence of categories used in the grammar. The feature section begins with the word ordering_features, followed by a list of features labels.

6 & 7 The boundaries section is optional. It gives the names of the boundary registers used in the grammar. The boundary section begins with the word boundaries followed by a list of boundary register labels.

10 & 11 The semrole section is optional3. It begins with the word semroles followed by a list of semrole labels.

12 & 13 The quants section is optional4. It begins with the word quants followed by a list of quant labels.

14 The default section is optional. It begins with the word default followed by the word cond, followed by a feature vector. The default section specifies the initial or default value for the condition vector used in the production specifica-

tions. Any feature value specified in the cond field of an individual production overrides the value in the default vector. The default section is intended to simplify condition vector specification.

15, 16, 17, 18 & 19 The macro section is optional. It begins with the word macros followed by the macro type and a series of macro labels and feature vectors. The macro labels begin with a '##' followed by a label when the macro is being defined while it begins with '#' when it is being used. The feature vectors begin with '+', '-', or '?' followed by a feature label or feature label..feature label. More than one feature label specification may be present within the feature vector. Macros which have already been defined may also be used in defining another macro. At present, only feature macros exist. Additional macro types will be added.

20 & 21 The production section is required. It begins with the word productions followed by a list of productions. There may be one or more productions in the grammar. Productions are the rules of the grammar.

22 Productions begin with the letter 'p' followed by the production label which is an alphanumeric string beginning with a letter. The label is followed by a lexical flag ('L', 'N', 'T'). This is optionally followed by morphology indication. The word cond appears next, followed by the condition feature vector. This is followed by the word change and the change feature vector. This is optionally followed by actions.

23, 24 & 25 Actions begin with the word action. This is followed by a series of action functions with their appropriate arguments.

26 & 27 A morphology indication consists of the word morph followed by a list of morphosyntactic properties labels.

28 & 29 Feature vectors consists of a series of feature values. These feature values can begin with a macro label or '+', '-', or '?' followed by feature label or feature label..feature label. Feature vectors indicate which features are "on", "off", or "don't care".

30 & 31 An agreement feature value set in the morph section is a list of feature value names between curly braces.

32, 33 & 34 An explicit agreement vector in the agree action is a list of agreement feature values between angle brackets. The agreement feature values may optionally have asterisks between them.

A.3 Grammar Semantics

2 & 3 The morphology section specifies both agreement feature values and independent morphological features. The agreement feature values are listed in sets inside curly braces.

15 - 19 Macros for feature vectors are implemented as ternary vectors. A list of macro labels and their corresponding ternary vectors are maintained locally for use in feat_vector.

15 A new list of macro labels and corresponding ternary vectors is created.

18 For each macro entry the macro label is saved and a ternary vector is created. After processing macro_vector, the ternary vector is saved.

19 For each feature label with a '+' indication, the corresponding position in the macro vector is set to ON. For each feature label with a '-' indication, the corresponding position in the macro vector is set to OFF. For each feature label with a '?' indication, the corresponding position in the macro vector is set to DON'T CARE. If a range of feature labels is given, the corresponding positions in the macro vector for the range of features are set according to the '+', '-', '?' indication. If an already defined macro is given, its vector changes the current macro's vector.

22 The lexical flag 'L' indicates a Lexical production (consumes a word), 'N' indicates a Non-lexical production (does not consume a word), 'S' indicates a Subcategory production (which does not consume a word but must be listed in the category list of the current lexical entry), and 'I' indicates the InitFinal production. The InitFinal's change vector is the initial state and its condition vector is the final state. There should be only one I production, which is considered Lexical. L is the default.

25 Action functions perform a variety of actions:

- Save saves the state in the boundary register.
- ShiftDown increments the Clause Level.
- ReturnUp decrements the Clause Level.
- FrontBegin records the Cursor position at FrontBegin.
- FrontHead records the Cursor position at FrontHead.
- FrontReset clears FrontBegin and FrontHead.

- Gramrole Assignment copies the gramrole value (must occur after a save action to the boundary with the same label as the lefthand gramrole).
- Agree checks agreement between two gramroles.
- Newref indexes the CURR gramrole to a new Reference.

28 & 29 If macros are used, the appropriate positions in the ternary vector are set according to the corresponding macro vector.

32, 33 & 34 Agreement vectors here are assembled into their internal bit-vector representation just as is done for the affix agreement vectors in the **paradigms** sections (see section 6 for details).

Appendix B
Lexicon Specification

B.1 Lexicon Specification Changes

The following lexicon specification, along with the syntactic and semantic description, is from [Owens 90]. I have updated it to include the changes needed for affix agreement processing. Rule 9 has been changed. Rules 18 and 19 have been added.

```
1     rvg_lexicon       ::=  macro_section morph_section
                             feat_section entries_section

2.1   macro_section     ::=  MACROS macro_type
2.2                     ::=  /\

3     macro_type        ::=  CAT macro_list

4.1   macro_list        ::=  macro_entry
4.2                     ::=  macro_list macro_entry

5     macro_entry       ::=  DEFMACRO macro_vector

6.1   macro_vector      ::=  NAME check_range
6.2                     ::=  MINUSPROD check_range
6.3                     ::=  MACRO
6.4                     ::=  macro_vector NAME check_range
6.5                     ::=  macro_vector MINUSPROD check_range
6.6                     ::=  macro_vecto MACRO

7.1   check_range       ::=  RANGE
7.2                     ::=  /\

8.1   morph_section     ::=  MORPHS morphology
8.2                     ::=  /\

9.1   morphology        ::=  morph_term
9.2                     ::=  morphology NAME
9.3                     ::=  morphology EWORD
9.4                     ::=  morphology CAT

10.1  feat_section      ::=  FEATURES feat_list
10.2                    ::=  /\
```

```
11.1 feat_list          ::=  NAME
11.2                    ::=  feat_list NAME

12   entries_section    ::=  ENTRIES entries_list

13.1 entries_list       ::=  entry
13.2                    ::=  entries_list entry

14   entry              ::=  EWORD NAME CAT cat_list wordpath

15.1 cat_list           ::=  NAME
15.2                    ::=  MACRO
15.3                    ::=  cat_list NAME
15.4                    ::=  cat_list MACRO

16.1 wordpath           ::=  MORPH NAME word_idiom
16.2                    ::=  /\

17.1 word_idiom         ::=  NAME word_idiom
17.2                    ::=  /\

18.1 morph_term         ::=  LBRACE agree_values RBRACE
18.2                    ::=  NAME

19.1 agree_values       ::=  NAME
19.2                    ::=  agree_values NAME
```

B.2 Lexicon Syntax

1 This is the outline of the lexicon specification. The lexicon specification consists of the following sections: macro section, morphology section, feature section, and entries section.

2, 3, 4, 5, 6 & 7 The macro section is optional. It begins with the word macros followed by the macro type and a series of macro labels and macro vectors. Macro labels begin with a '##' followed immediately by a label when the macro is being defined, while it begins with a '#' when it is being used. Macro vectors begins with a production label or '-' followed immediately by a production label or production label..production label. More than one production label

specification may be present within the macro vector. Previously defined macros may also be used. At present, only category vector macros are implemented. Other macro types will be added later.

8 & 9 The morphology section is optional. It begins with the word morphosyntactic_properties. The format of the morphology section here is identical to the one in the grammar source file.

10 & 11 The features section is optional. It begins with the word features followed by a list of feature labels.

12 & 13 The entries section is required. It begins with the word entries followed by a list of lexical entries.

14 Lexical entries are required. A lexical entry begins with the letter 'e' followed by the lexical label (word or formation). The word cat appears next followed by a category list. This is optional followed by wordpath.

15 A category list is a list of category labels and/or macro labels.

16 & 17 Wordpath begins with the word morph followed by a morphology specification.

B.3 Lexicon Semantics

1 Prior to processing any of the sections in the lexicon specification, a new lexicon is created, the grammar is read in from synindex.rvg, and all production labels are obtained. After all sections have been processed, the lexicon is stored in lexicon.rvg.

2 - 7 Macros for category vectors are implemented as byte vectors. The production index for each "on" production is added to the byte vector. A dictionary of macro labels and their accompanying byte vectors are maintained locally for use in cat_list.

3 A new list of macro labels and corresponding byte vectors is created. This list is converted into a dictionary after all macros of this type are defined.

5 For each macro entry, the macro label is saved and a byte vector is created. After processing each entry, the byte vector and label are stored in a set.

6 & 7 For each production listed without a '-' indication, a byte is set in the byte vector. For each production with a '-' indication, the byte is removed from the byte vector if it is there. A check is made to insure that the production labels actually exist. For each macro label, the macro's byte vector is added to the current byte vector.

9 The agreement feature values specified in the morphology section should match exactly with those specified in the morphology section of the grammar source file.

Biography

The author, Edward James Labuda, was born on February 6, 1961 in Summit, New Jersey to Mr. Edward F. Labuda and Mrs. Gayle G. Labuda. Mr. Labuda received a B.S. degree in computer science from Yale University in 1983. From 1983 to 1989 he was employed by American Telephone & Telegraph in Allentown, Pennsylvania as an applications software developer for a large factory-floor tracking and control system. Since 1989 he has worked for AT&T in Piscataway, New Jersey as an in-house consultant on artificial intelligence technology. Mr. Labuda will receive the M.S. degree in computer science from Lehigh University in June 1990.